CYBER-AWARENESS IMPROVEMENT USING ARTIFICIAL INTELLIGENCE TECHNIQUES

Soorena Merat and Wahab Almuhtadi
(1) SCE Consulting Inc., (2) Algonquin College, Ottawa, Ontario, Canada

Submitted: Dec. 15, 2014 ***Accepted: Jan. 24, 2015*** ***Published: Mar. 1, 2015***

Abstract– The primary contribution of this experiment is the development of a framework on which a variety of multitasking processes can be mapped. A software model named SHOWAN is developed to represent, capture and learn the cyber awareness behavior of a computer process against multiple concurrent threads. At the beginning of the experiment, the examined process outperformed, and tended to manage numerous tasks poorly, but it gradually learned to handle fewer tasks and excelling in each of them.

Index Terms– Cyber Security, Queuing Management, Task Prioritization, Normative Model, Cyber Multitasking Performance, Non-maskable task.

I. INTRODUCTION

The main cause of cyber-attack is secured software system error instead of other causes such as hardware failures. Many organizations have studied cyber-attacks and have discovered repeated occurrences of task management errors in which the soft process chose to utilize the wrong thread. These errors in the layers of processing can cause firewall and other security software malfunctions, and have disastrous consequences for highly protected environments such as nuclear power plants. When a process force with several threads occurs the interrupt handler may mistakenly attend to the non-maskable, but lower priority task, or exhibit a poor queue management strategy. The question of how well a process can manage multiple concurrent threads, while also performing high quality traffic tasks such as data transfer with very large amount of data is the main consideration. The primary concern is to have high security while minimizing the delay. In order to achieve this, the process is required to have an efficient feedback control which adapts the idle time to learn traffic change dynamics by constraining the queue length at a predetermined value.

Most of the existing studies focus on models based on estimation and control theories. In these models, future process loads and traffics are estimated to find how to reach a desired state and their efficiency is ensured to optimize certain criteria. However, such methods and their subsequent models are generally in the continuous and sequential control task traffic mode which is not the focus of the current experiment. In this experiment, it is assumed that as soon as the process load/traffic manager, starts learning and acting, it progresses towards the desired status at a constant rate, without optimization involvement. Therefore, the prime interest is whether a high index/load process thread can be considered to be non-maskable at any point of time, even if it causes a dangerous situation, or if it can be simply ignored. This aspect of cyber security falls into the category of multitasking and load management control.

This experiment considers the load management routine as a single task and it uses a selective task scheduling approach, which allows for a learning system based on algorithmic comparison and pre-emptive task selection.

II. ALGORYTHM PRINCIPLE

The analogy of SHOWAN (is Kurdish for shepherd) is introduced to describe a multitask initiative used to capture upon multiple concurrent threads. A functional representation of multitasking is the shepherd which has several tasks in a queue, and makes sure all of them are engaged to secure the flock against

possible attacks. During a combined life-cycle, shepherd can never assume that the tasks are completed, and therefore, it requires continuous attention; the theory being that the more attention paid, the safer the environment. Consequently, performance metrics cannot be measured by the number of safely completed threads. Instead, the subject of interest would be the number of managed high priority and high index tasks on average and over a defined period of time. If different importance values (priorities) are assigned to individual tasks (including very high priority threads), the average safe thread passes over time (with minimum number of unmanaged threads) would be a good provides an accurate and pertinent performance measure.

An interactive, graphical, user software is developed with the objective function of improving the learning curve of the computer process to an average satisfaction level on high priority tasks over time. This cognition improvement software uses memory training concepts that enhance artificial intelligence and memory by using rhyming and number placement techniques [5]. To start the experiment, a very common type of thread pool selected, which is fixed thread pool. This type of pool always has a specified number of threads running; if a thread is somehow terminated while it is still in use, it is automatically replaced with a new thread. Attacks are submitted to the pool via an internal queue, which holds manually created handler and drops extra tasks whenever there are more active tasks than threads. All of the running threads executor and implementations in java use thread pools concurrently, which consist of working threads. This kind of thread exists separately from the Runnable and Callable tasks it executes and is often used to execute multiple tasks. The reason is to capture the thread behaviors and reaction to attacks while on idle. A manually simulated thread with variable Load/Priority index is introduced to each thread within the nominated process. The attacks initiate the start of the artificial intelligence script, which limits the thread's load priority index to a maximum of 90% and a minimum of 10% (Figure 1). The attacks are implemented using fork/join framework is an implementation of the Executor Service interface that helps take advantage of multiple soft processors. A very common concept has been considered to develop attacks statement, which is concurrent ability to execute threads in timely manner and is known as thread/attack liveliness within two categories:

1. Intrinsically locked to thread sequences.
2. Multiple thread synchronized.

Each thread is designed with ability to be broken into smaller pieces in recursive periods. The goal is to use all the available processing power to enhance the performance of your application. The margin is an indication of CPU/Process load factor, which normalizes other threads' activity and prevents the result of reaching zero. It is selected to avoid 100% load as this is too high an index. However, for every time period that a high priority task pass the high/low margin, a penalty of a factor of 20% of the task's weight is deducted [2]. The choice of the penalty factor is valuable, as too low a factor could be flagged as it is at the zero index for a significant period of time. The optimal solution to the model is a sequence of high index threads (attacks) that has to be attended to or managed at every time period throughout the planned zone in order to maximize the objective function.

III. THE METHOD

For this experiment, a sample data transfer process was selected under the DSDV routing protocol test without mention of a firewall. The exclusion of a firewall is due to the fact that an attack (Passive/Active) can be initiated by an entity inside the security perimeter.

The core of this experiment is a program which implements the DSDV routing protocol. It has been used to transfer over a path between two nodes as well as handle changes in the routing table, including the link break. The program includes three threads, one for sending messages, one for receiving messages and one for main program, which creates other two threads.

The sending thread will check the batch files and update the routing table of the node, transfer it into a piece of message and broadcast it. It will also increase the sequence number of this node every few seconds. The receiving thread will always wait for messages from other nodes, and update the routing table once such a message received. A lock mechanism is used in this program to make sure only one thread would update the routing table in one time.

DSDV starts out high and then drops over the first 100 numbers, because the connections are starting up over the first 100 s, so the load on the network is constantly increasing over that time by automatic scripts, which check the transients and passages throughout the process. (Figure-1)

We assumed that the route congestion is presumably not so severe. So that DSDV can track routes to all destinations and DSDV nodes are not dropping packets in the RTR level. The reason is that the route tuning practices to make a congested network run better and to bring the network slightly out of congestion collapse such as using shorter IFQs parameters is not the purpose of this experiment.

Before start the simulation, we create four template TCL scripts to be used by our batch file to manually simulate attack scenarios. Batch file runs the simulation based on the test scenario varying speed and pause time. Batch files also copy the scenario based on the manually initiated threads, to run the AWK and recreate the LOG and finally trace and record batches will conclude the decision process to ignore or end the script. This whole process will give DSDV enough time to continuously update the entire routing table periodically and as per request, which creates a slight delay in delivery, but the end to end delay does not change to increase in the number of attacks and it only may increase number of hops.

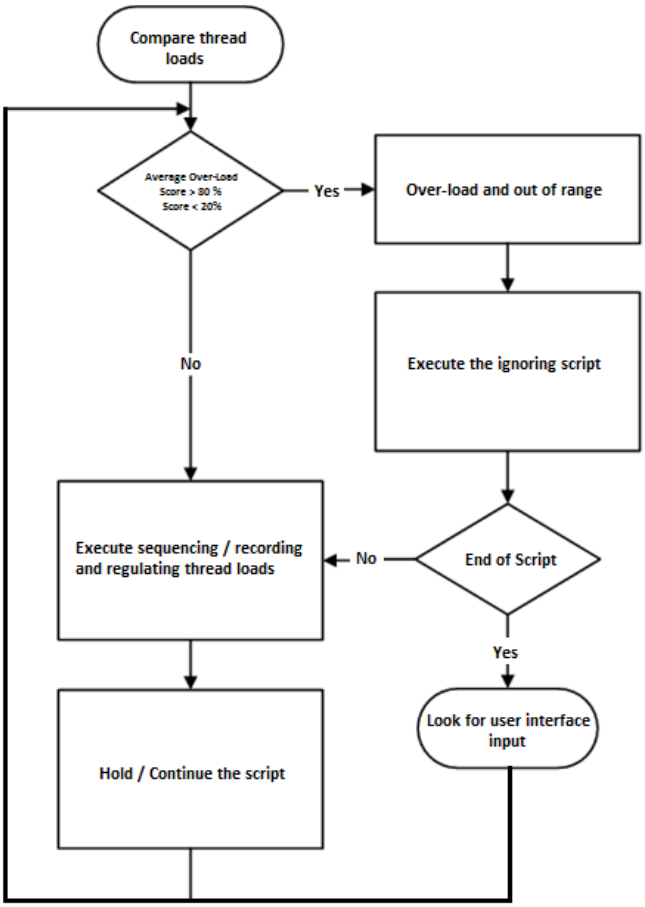

Figure 1: The Proposed Algorithm

In addition, ten threads at different load and priority levels were initiated. [3] A low-fidelity multitasking software environment with graphical user interaction named SHOWAN was developed and used. In this environment, a user identified attack could be simulated and the behavior of the transfer process (task acceptance and ignorance rates from their satisfactory status) was measured. Task related factors such as task priority, penalty coefficient, and the duration of the experiment could be manipulated.

We have used AWK scripting language and Java code for extracting data from trace files. The analysis is done for generating various performance metrics like packet delivery fraction, average end-to-end delay, packet loss, packet delay, routing overhead and route acquisition time [18].

IV.THE SEQUENCE

The sequence of generating word placement and rhyming model starts from sampling and extracting data from each channel under attack, which creates pool of valid alternatives. This first step in sequence is to sample and expand data in the input prose with meaning-equivalent alternatives. The next phase is to measure the compared samples and collect, since different data strings have different meter constraints [20]. For example, a passive cyber-attack initiated by an entity inside security perimeter should follow a specific prose, which represents a natural flow of samples. An example cyber security initiated by an entity outside of security perimeter, may follow an iambic pentameter constraints, implying a rhythm that words establish in a line. This method introduce a combination of 10 weak/strong sample pairs per line which creates a string under the required constraints using choices from the first phase.

Providing the pre-calculated measures from previously recorded cyber-attacks (Original-attack) simplifies the look-up task, in which, we collect rhyme information of each sample in the original-attack or expanded sample set. The junction of the rhyming sets across different data sets is taken to find rhyming pairs of samples. Using the expanded word set greatly improves our chances of finding valid matching rhyme pairs.

This experiment annotates the risk categories according to scores. There are 5 score classifiers in the project corresponding to:

- Threads passing higher margin scores that might contain high risk
- Threads passing higher margin scores that sure contain high risk
- Threads that stay in safe operation area
- Threads passing lower margin scores that sure contain high risk
- Threads passing lower margin scores that might contain high risk

Rhyming and placement method requires to use known category along with the others, otherwise AI script classifier would be able to identify only scores with every given threads. But in real world

there are other types of threads that do not fall into either of the above presented categories, so it is required to meddle with AI classifier thresholds. The graphical software user interface mimics threads load and priority indexes, which were shown as graphical monitors. (Figure-2)

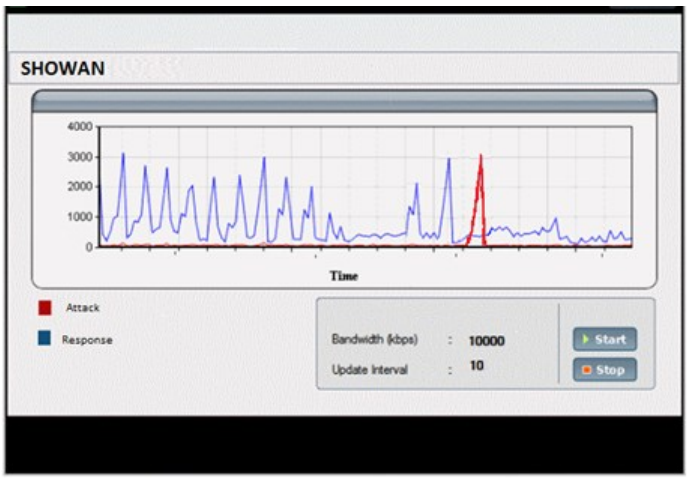

Figure 2: Initial Reaction To A High Risk Activity Captured By SHOWAN

As the process runs, the load index indicators drop, demonstrating how the system resource usage becomes normalized over time. The software updated the status of the threads every tenth of a second. Therefore, the process could, theoretically, react to a sudden load unbalancing of any of the threads as fast as tenth of a second, although it is longer due to real world processor-load limitations.

We also assume only one routing protocol is necessary for confirming no more than one connection to the event. Furthermore, the software calculates changes in the measures of an unsuccessful attack as the duration of the attack minimizes [1].

The software computes a numerical score for each thread, depending on how normal the process can keep the average status of the threads over time, how long threads were indexed in the 'out of the range zone', and whether or not the thread with a higher index value crashed. (Figure-3)

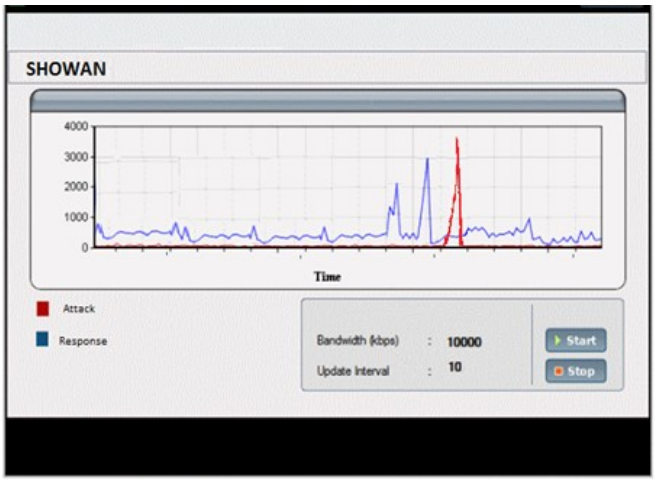

Figure 3: Normalized Response To A High Risk Activity Captured By SHOWAN

The software recorded which thread was more affected at any point in time. The process behavior in response to the simulated attack was then replayed on the computer and studied in order to understand the process stream line around denial of service (DOS) or R2L [10]. This helped to discover how the process handled the surprises against familiar threads. Figure 3 shows a sample of the graphs generated.

V. EXPERIMENTAL PROCEDURE

The procedure involved running the software multiple times and under different scenarios in order for the examinee process cycles to learn and capture a normal range of scores within nominated strategies. It does not model all the details and characteristics of an attack but obtains simple models where multiple attack scenarios can be observed. The scenarios differ from each other in terms of index and weighted value. The length of the test for each scenario is ten to fifteen minutes, during which the graph for each process/thread to scenarios is recorded. In all cycles, scenarios started with an initial index of 20%, and penalties of 30% of the index value were deducted for every time that they neared upper and higher limits. The primary intent of the scenarios was to discover how the cognitive act of the computer's process can be improved by learning the normal process stream line while effectively ignoring the irregular activities. Study of the process behavior also indicates how close to failure a complex process reached when the complexity of the attack increased. [3]

We used the random waypoint model where each thread starts the transmission simulation by remaining stationary for minimum pause times. It then selects a random destination and moves toward that, assuming at maximum speed. Upon reaching the destination node, the thread pauses again, selects another destination, and proceeds there as previously explained, repeating this behavior for the duration of the simulation. It has been considered to use the advantage of multicore processing, in order to speed-up of the script execution, but using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

To really take advantage of a multi-core system, the script should split up the main processing into multiple threads as well and requires an algorithm, capable to be parallelized. This is only possible to a point however, depending on the algorithm. A simple explanation would be:

For a

$$\boldsymbol{X = G + H + I + J}$$

To be parallelized in a most simple way (Would be more complex under a CPU thread management system):

$$\boldsymbol{T_{0=}T_{G+}T_{H}}$$

And:

$$\boldsymbol{T_{1=}T_{I+}T_{J}}$$

So:

$$\boldsymbol{X = T_0 + T_1}$$

T_0 and T_1 could be calculated in parallel threads. However, to calculate X, we need the results of both threads. So most part of the algorithm execution must be in parallel, but another part is implicitly sequential. It depends on results from earlier part of calculations, so there is no way to run this calculation in parallel with other dependent calculations [19].

The peak performance of the processor can be calculated by multiplying the frequency by the number of cores and the number of instructions that can be issued for each core. Assuming that the single core processors runs at 1.4GHz, then the processor can sustain 1.4*1 = 1.4 billion instructions per second.

Because multiple threads are sharing a single core, the question of whether the single core is fully loaded or not becomes interesting. For example, suppose that the core is fully loaded. That means each thread should be getting its fair share of the available instruction slots. So each thread should be able to issue an instruction every few cycles.

In this case it becomes interesting to ask whether running fewer threads on the core would improve the latency of the application. Or while CPU is fully loaded, were the new attacks would be properly handled. Alternatively, if not all the threads on a particular core are busy, it is interesting to ask whether the core has sufficient instruction issue capacity to handle additional threads.

In this experiment, assumption has been made that utilisation of single core CPU is already optimized and manual experimental attacks are not to determine whether fewer number of threads would help CPU performance to benefit from fewer or more virtual processors being assigned work. We have also estimated the CPU resource consumption as following sample, in order to be used during cyber-attacks analysis, (Figure-4):

Counter	Comment	Total in Cycle	Time at 1.4GHz	% of Total Runtime
SATB-COMP-FULL	Number of Cycles when stored Attack buffer is full	11	0	2
PAFP-instr-cnt	Attack Floating-point instruction count	23	0	0
IAC_miss	Instruction Attack cache miss	21	0	1
ADC_miss	Attack Data cache miss	2	0	0
ATLB_miss	Attack Instruction TLB miss	85	0	0
DTLB_miss	Data TLB miss	95	0	0

Figure 4: CPU Resource Consumption Estimation During Cycles

VI. THE RESULTS and CONCLUSION

The results of the experiments were compared through graphical indicators for two types of attack (Locked Sequence and Synchronized), during qualitative comparison. It indicates the best, worst, and mean performance reaction of the examined thread. [6]
For quantitative comparison, the process overreacted to the high index and high load threads in early implementation phases. It was not able to disengage the synchronized threads and attempted to handle too many, causing poor performance and out of margin penalty. This effectively reduced switching time to zero. Even with manual manipulation assistance, none of the threads were able to beat the high margin. After a number of successful attempts, the process has improved and learned the normal operation load/priority indexing routine. As a result of the learning relaxation, the maximum best score range was scaled within 56% to 85% of the near-optimal for all different attacks. The overreaction, caused the process to ignore threads with least priority index or the lower value threads had a much better performance [14].

During the process, some standard rhyming detection method has been called, while looking up the manually developed multitasking attacks. The purpose is to increase machine learning capabilities and help computer system administrators to learn the attack symptoms from machine-learning prospective. The methods, presented with an interface showing the original samples and manipulating elements. An algorithm dynamically compares the composed manipulating elements and finally let the administrator personnel to provide some alternative combination and understand the safe operating margins.

For the process to learn good behavior, selection, and the number of threads to which, it must react is much more important than the specific thread’s moment-to moment index changes (Figure-5).

There are several benchmarking algorithm based on JAVA available on the market, Many of which are focused on applet performance of hyper-threading. Some others test the threaded performance under specific conditions; however, they don't allow to customize and focus on specific individual operations that could affect performance of the operation.

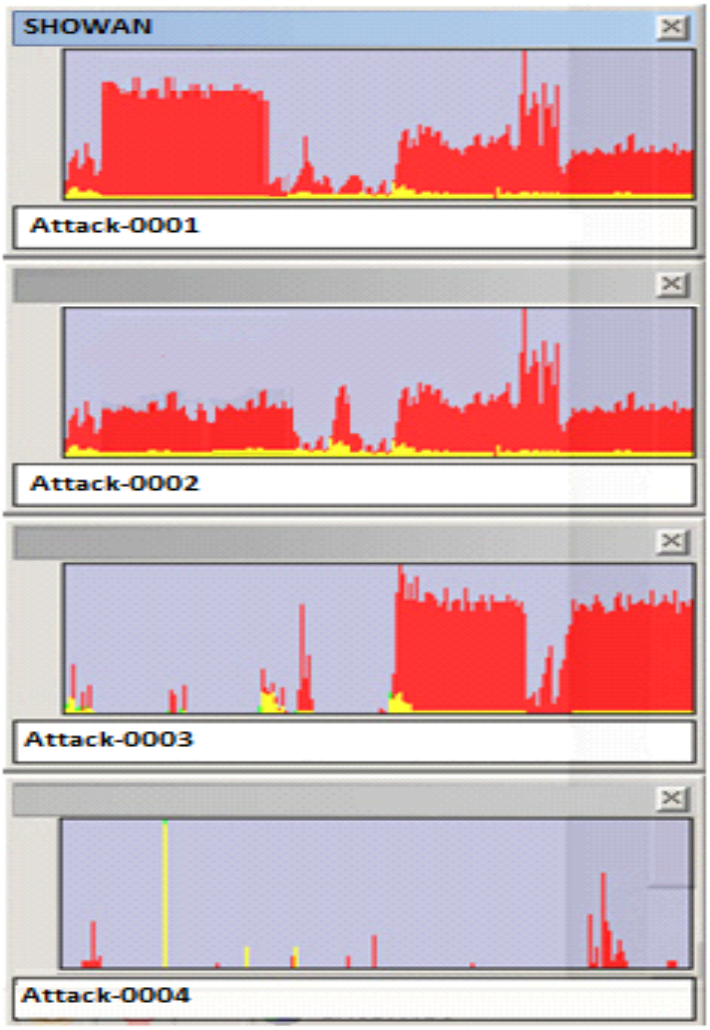

Figure5: Comparison of Few Attacks (Route Efficiency Metric vs. Node Congestion Metric)

This experimental study is not similar to performance monitor counters, which are basically hardware registers that measure events occurring in the processor. They normally can be used to help find performance bottlenecks by identifying an excessive amount of events of a particular type. For example, a high number of conditional branch instructions may indicate a section of logic that, if rearranged, might lower the number of branches required. Even though performance counter can bring these issues to light, it is up to user to match them to application code and decide how they will help you improve application performance.

The present experiment provides a graphical representation of each thread's state at a particular time in the run time. Each thread state is color-coded to help identifying what each thread is doing in conjunction with all of the others. Threads that go through multiple state changes are easily identifiable through the changing colors in the track pane. Figure 10-2 shows fifteen threads being tracked.

A cyber-attack assumes to start as a multi-thread or hyper-thread operation, which may synchronously activates some processor's thread and do not duplicate all available resources. The idea behind this experiment is that, while two threads performing fundamentally similar operations on separate logical processors, the equal load observation will likely see little performance difference and gain lose.

Technically for a multi-thread process to be a benefit; the two threads coexisting on a physical CPU must perform a variety of operations to allow the processor to make better use of latency and other similar factors. The test bed used in this experiment are by no means to benchmark processor architecture or design. The goal is to stress the processor, using different processor resources, and try to gain detail insights into the effects of unbalanced and non-rhythmic multi-processing. De-noising is also an important part of post result processing. For example several small pieces of a captured result is manipulated and stained by noise points and direct de-noising may leave scratch and traces. Using over-scaling to 350% normalize and transforms the characteristics of the signal as it appears in figure -6.

Every round of optimization, results decompose and expand into more sub-sections by SHOWAN, of which corresponds to a smoothed version, and the other corresponds to details versions.

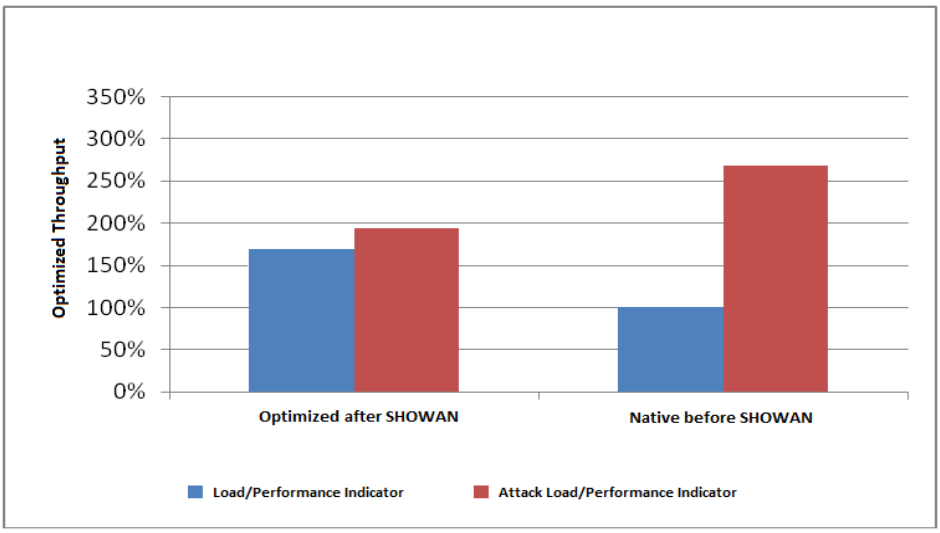

Figure 6: Comparison of Optimized vs. Non-Optimized Load/Performance Metrics

While the process is on idle, SHOWAN still runs in the background managing and analyzing the major initiatives taken by a process against higher index tasks on behalf of process threads, involving resources and performance in internal and external environments in previous scenarios. This idle processing improves the process learning curve. Once the thread is flagged by the process, it will handle much faster and efficiently in subsequent attempts. This means, strategic task management plays a major role and is more important than tactical task management.

VII. REFERENCES

[1] Neal Patwari, A. O. Hero III, M. Perkins, N. S. Correal, and R. O'Dea, "Relative location estimation in wireless sensor networks", IEEE Transactions and Signal Processing, Vol: 51, Issue: 8, 2003-8, Pages: 2137–2148.

[2] Jonathan Hui and David Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale", In SenSys '04, ACM Conference on Embedded Networked Sensor Systems November 3-5, 2004. Baltimore, MD, In Proceedings of the 6th International Conference on Embedded Networked Sensor Systems.

[3] Erika Greene, Tugba Bodrumlu, and Kevin Knight, "Automatic analysis of rhythmic poetry with applications to generation and translation." In Proceedings of EMNLP, Conference on Empirical Methods in Natural Language Processing, October 9-11, 2010, MIT, Massachusetts, USA.

[4] Diana McCarthy, and Roberto Navigli, "Semeval-2007 task 10: English lexical substitution task." In Proceedings of the 4th International Workshop on Semantic Evaluations, Pages 48 to 53, 2007.

[5] Erika Odilia Flores Popoca, Maximino Miranda García, Socorro Romero Figueroa, Aurelio Mendoza Medellín, Horacio Sandoval Trujillo, Hilda Victoria Silva Rojas, Ninfa Ramírez Durán. "Pantoea agglomerans in Immunodeficient Patients with Different Respiratory Symptoms." The Scientific World Journal 2012, Article ID 156827, 8 pages.

[6] Christian Kehlmaier, Radek Michalko, Stanislav Korenko, "Ogcodes fumatus (Diptera: Acroceridae) Reared from Philodromus cespitum (Araneae: Philodromidae), and First Evidence of Wolbachia Alphaproteobacteria in Acroceridae, Annales Zoologici, 2012, 62:2, 281-286.

[7] M. Verbeek, A. M. Dullemans, P. J. van Bekkum, R. A. A. van der Vlugt, "Evidence for Lettuce big-vein associated virus as the causal agent of a syndrome of necrotic rings and spots in lettuce." Plant Pathology, Volume 62, Issue 2, pages 444–451, April 2013.

[8] Zhiwen Wang, Neil Hobson, Leonardo Galindo, Shilin Zhu, Daihu Shi, Joshua McDill, Linfeng Yang, Simon Hawkins, Godfrey Neutelings, Raju Datla, Georgina Lambert, David W. Galbraith, Christopher J. Grassa, Armando Geraldes, Quentin C. Cronk, Christopher Cullis, Prasanta K. Dash, Polumetla A. Kumar, Sylvie Cloutier, Andrew G. Sharpe, Gane K.-S. Wong, Jun Wang, Michael K. Deyholos, "The genome of flax (Linum usitatissimum) assembled de novo from short shotgun sequence reads." The Plant Journal, 2012 Nov;72(3), P: 461-73.

[9] Raffaele Ronca, Michalis Kotsyfakis, Fabrizio Lombardo, Cinzia Rizzo, Chiara Currà, MartaPonzi, Gabriella Fiorentino, Josè M.C. Ribeiro, Bruno Arcà, "The Anopheles gambiae cE5, a tight- and fast-binding thrombin inhibitor with post-transcriptionally regulated salivary restricted expression", Insect Biochemistry and Molecular Biology, 2012, 42:9, 610-620.

[10] N.K. Suryadevara and S.C. Mukhopadhyay, "Wireless Sensor Network Based Home Monitoring System for Wellness Determination of Elderly", IEEE Sensors Journal, Vol. 12, No. 6, June 2012, pp. 1965-1972.

[11] Benjamin A. Sandkam, Jeffrey B. Joy, Corey T. Watson, Pablo Gonzalez-Bendiksen, CaitlinR. Gabor, Felix Breden, "Hybridization Leads to Sensory Repertoire Expansion in A Gynogenic Fish, The Amazon Molly (Poecilia Formosa): A Test Of The Hybrid-Sensory Expansion Hypothesis", Evolution, Volume 67, Issue 1, pages 120–130, January 2013.

[12] Tatiana V. Danilova, Bernd Friebe, Bikram S. Gill, "Single-copy gene fluorescence in situ hybridization and genome analysis, loci mark evolutionary chromosomal rearrangements in wheat chromosome, Theoretical and Applied Genetics, Volume 127, issue 3, PP 715-730.

[13] Nourou S. Yorou, Moussa Diabate, Reinhard Agerer, "Phylogenetic placement andanatomical characterisation of two new West African Tomentella (Basidiomycota, Fungi) species." Mycological Progress, Volume 11, Issue 1, PP 171-180, February 2012.

[14] Joseph Polastre, Robert Szewczyk, Cory Sharp, and David Culler, "The Mote Revolution: Low Power Wireless Sensor Network Devices", Proceedings of IEEE Hot Chips 16, August 22-24, 2004. Palo Alto, CA.

[15] David Robert, Peter Schmitt, Alex Olwal, MIT Media Lab, "Cloud Rhymer: Prototype Demo and Intervention Proposal", Proceedings of the 12th International Conference on Interaction Design and Children, Pages 507-510.

[16] Nik Noordini Nik Abd Malik, Mazlina Esa, Sharifah Kamilah Syed Yusof, and Jayaseelan Marimuthu, "Optimizing of Node Coordination in Wireless Sensor Network", Asia Pacific Microwave Conference, 7-10 Dec 2009, Pages: 2336-2339.

[17] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications", International Conference for High Performance Computing, Networking, Storage and Analysis, Super Computing conference 2012, Article Number 51.

[18] N.K.Suryadevara, A. Gaddam, R.K.Rayudu and S.C. Mukhopadhyay, "Wireless Sensors Network based safe Home to care Elderly People: Behaviour Detection", Sens. Actuators A: Phys. (2012), doi:10.1016/j.sna.2012.03.020, Volume 186, 2012, pp. 277 – 283.

[19] A. Gupta, J. Sampson, and M. B. Taylor. Timecube, "A manycore embedded processor with interference-agnostic progress tracking", International Conference on Embedded Computer Systems Architectures, Modeling and Simulation, 15-18 July 2013, Pages 227 – 236.

[20] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta, Quantifying effects of shared on-chip resource interference for consolidated virtual machines, Applied Computing Symposium, 2nd Symposium on Cloud Computing, 2011. Article No. 22.

[21] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications", 38th Annual International Symposium on Computer Architecture, June 4-8, 2011, Pages 283-294.